2020년도 2학기 자료구조
HW2 결과보고서

20190345 김동현

1. 과제 요약

1번 문제는 2글자 이상, 30글자 이하의 영어단어를 입력 받은 뒤, 연속적으로 중복이 발생하는 글자를 제거한 문자열을 출력하는 문제이다. 2번 문제는 KMP 알고리즘을 수정하여 1글자 이상 30글자 이하의 문자열과 패턴을 입력 받은 후, 매칭되는 부분을 삭제하고 남은 문자열을 출력하는 문제이다. 3번 문제는 1글자 이상 10글자 이하의 문자열과 패턴을 입력받은 뒤, 패턴의 anagram과 문자열이 일치하는 문자열의 인덱스를 출력하는 문제이다.

2. 과제 환경

CSPRO 환경에서 실습을 진행하였다.
CSPRO 서버 : cspro.sogang.ac.kr

3. 과제 구현 내용

1번 문제는 문자열 입력 부분과 중복을 제거하는 부분, 출력 부분으로 이루어진다. 문자열은 문자형 배열 a에 저장하며, 중복된 결과를 저장하는 배열은 b이다. 문자열 입력 부분은 다음과 같다.

```
gets(a, 100);
if (strlen(a) < 2 || strlen(a) > 30) {
        fprintf(stderr, "size error");
        return 1;
}
```

입력받고자 하는 문자열을 gets를 통해 입력받는다. 이후 입력받은 문자열의 길이가 2글자 미만이거나, 30글자를 초과할 경우 "size error"를 출력한 뒤, 종료한다.
연속된 중복 글자를 제거하는 부분은 다음과 같다.

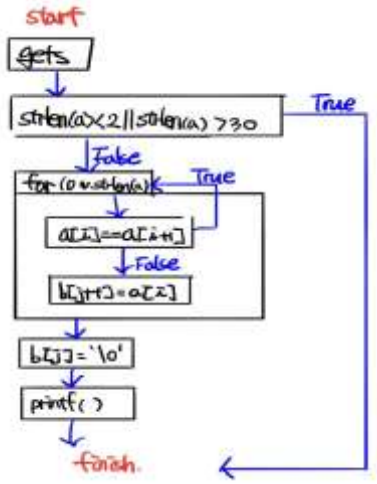
```
for (i = 0; i < strlen(a); i++) {
        if (a[i] == a[i + 1]) {
                continue;
        }
        b[j++] = a[i];
}
        b[j] = '₩0';
```

문자열을 순차적으로 한글자씩 접근한다. 만약 현재 접근한 글자가 다음 글자와 같다면 이는 중복된 글자이므로 아무런 작업을 수행하지 않고 다음 글자로 넘어간다. 만약 중복된 글자의 마지

막 부분에 접근한다면 마지막 글자는 그 다음 글자와 다르므로 b배열에 추가된다. a배열을 모두 검사한다면, b배열에는 중복이 제거된 문자열이 입력되어 있고, 마지막에 EOS를 의미하는 '₩0'을 추가해 문자열로 만들어준다.
출력부분은 문자열을 저장하는 문자 배열 b를 출력해준다.

```
printf("%s₩n", b);
```

1번 문제의 구조도는 다음과 같다.



2번 문제는 문자열과 패턴을 입력받는 부분, 문자열과 패턴이 일치하는 부분을 찾아 문자열에서 제거하는 부분, 삭제된 결과 문자열을 출력하는 부분으로 구성된다. 문자열은 string에 패턴은 pattern 배열에 저장되며, KMP 알고리즘을 위한 failure 배열을 위한 배열 포인터와 문자열에서 일치하는 패턴을 삭제한 문자열을 입력받을 배열의 포인터 result를 선언하였다. 다음과 같이 문자열과 패턴을 입력받는다.

```
gets(string, 100);
gets(pattern, 100);
if (strlen(string) > 30 || strlen(string) < 1) {
        fprintf(stderr, "string size error");
        return 1;
}
if (strlen(pattern) > 30 || strlen(pattern) < 1) {
        fprintf(stderr, "pattern size error");
        return 1;
}
if (strlen(string) < strlen(pattern)) {
        fprintf(stderr, "string is shorter than patternr");
        return 1;
}
```

문자열과 패턴을 gets 함수를 이용해 입력 받은 뒤, 문자열과 패턴이 2글자 이상, 30글자 이하가 되는지 검사한다. 또한 문자열의 길이가 패턴의 길이보다 길거나 같은지 검사한다.

이후 KMP 알고리즘을 위한 failure 배열과, 결과값을 저장할 result 배열을 동적할당한다. 각각의 배열은 패턴의 길이만큼 정수형 배열로 동적할당, 문자열의 길이만큼 문자열 배열로 동적할당 한다. 이후 문자열과 패턴을 검사하고 일치하는 부분은 삭제하는 함수 pmatch는 다음과 같다.

```
void pmatch(char *string, char *pat, int failure[], char *result){
	int i=0,j=0,index,correct=0;
	int* match;
	int lens=strlen(string);
	int lenp=strlen(pat);
	match = (int*)calloc(strlen(string), sizeof(int));

	do {
		i = correct;
		j=0;
		while (i < lens && j < lenp) {
			if (string[i] == pat[j]) {
				i++;
				j++;
			}
			else if (j == 0) {
				i++;
			}
			else {
				j = failure[j - 1] + 1;
			}
		}
		index = ((j == lenp) ? (i - lenp) : -1);
		if (index == -1) {
			correct++;
		}
		else {
			for (i = 0; i < strlen(pat); i++) {
				match[index + i] = 1;
			}
			correct = index + 1;
		}
	} while (correct < strlen(string) - strlen(pat));

	j = 0;
	for (i = 0; i < strlen(string); i++) {
```

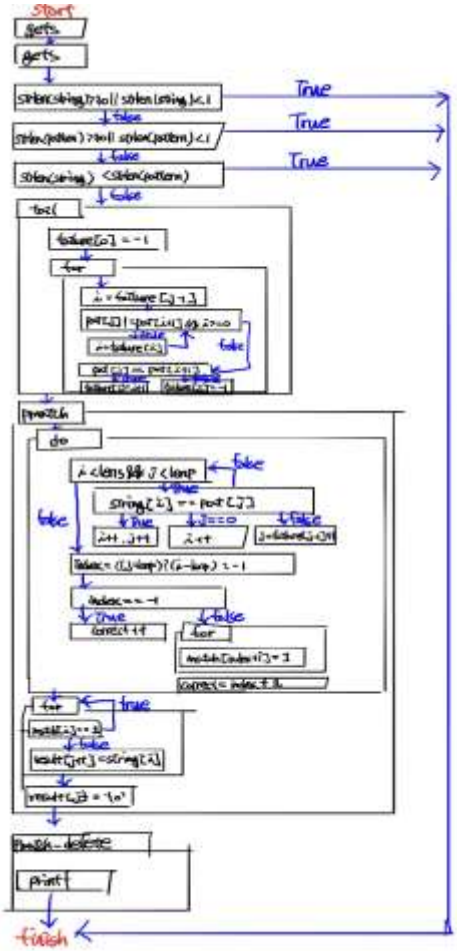
```
                if (match[i] == 1) {
                        continue;
                }
                result[j++] = string[i];

        }
        result[j] = '₩0';

        free(match);
}
```

우선 do-while 문은 KMP 알고리즘을 수행하는 부분이다. 문자열의 처음부터 한글자씩 뒤로 밀려가며 일치하는지 검색한다. 이때 KMP 알고리즘의 결과값은 index 변수에 저장되며 일치하지 않을 경우 -1을 일치하는 경우 해당 문자열의 시작 인덱스를 반환한다. Correct 변수는 현재 문자열을 검색하는 문자열의 시작 부분 인덱스을 의미한다. 따라서 검사가 한 번 완료된 후에는 다음 위치로 변경해주어야 한다. 만약 결과값이 -1이면 correct 값을 1 증가하여 다음 위치의 문자부터 검색을 시작한다. 만약 결과값이 반복되는 문자열의 인덱스일 경우에는 해당 인덱스의 다음 부분으로 correct를 변경한다. 이후 패턴과 문자열이 일치하는지 여부를 보여주는 matching 배열의 값을 1로 변경한다. Matching 배열이 의미하는 바는 문자열의 각 부분이 패턴과 일치했는지 여부를 보여주며 1이면 일치, 0이면 불일치를 의미한다. 이후 matching 배열이 0인 부분의 인덱스 위치의 문자열의 문자 값을 result 배열에 순서대로 추가한다. 마지막에는 EOS를 의미하는 '₩0'을 추가한다.

삭제된 문자열 result를 출력하는 함수 pmatch_delete는 인자로 result의 포인터를 넘겨준다. 이후 문자열을 출력한다.

```
void pmatch_delete(char *result) {
        printf("%s₩n", result);
}
```

문제2번의 구조도는 다음과 같다.



3번 문제는 문자열과 패턴을 입력받는 부분, 패턴의 anagram과 문자열이 일치를 검사하는 부분, 일치하는 부분의 인덱스 값을 저장한 배열을 출력하는 부분으로 구성된다. 문자열과 패턴을 입력받는 부분은 다음과 같다.

```
gets(string, 100);
gets(pattern, 100);

if(strlen(string)>10||strlen(string)<1){
        fprintf(stderr, "string size error");
        return 1;
}
if (strlen(pattern) > 10 || strlen(pattern) < 1) {
        fprintf(stderr, "pattern size error");
        return 1;
}
if (strlen(string) < strlen(pattern)) {
        fprintf(stderr, "string is shorter than pattern");
        return 1;
}
```

문자열과 패턴을 gets로 입력받는다. 이후 문자열과 패턴의 길이가 1글자 이상, 10글자 이하인지 확인하고, 문자열의 길이가 패턴의 길이보다 크거나 같은지 확인한다. 이후 패턴의 anagram을 확인하는 방법은 다음과 같다. Anagram은 구성하고 있는 문자의 순서를 바꾼 것이다. 순서에 상관없이 패턴과 문자열이 1대1 대응을 만족하면 해당 문자열은 패턴의 anagram이 될 수 있다. 이러한 검사를 위해 일대일 대응관계를 보여주는 matching 배열과 anagram이 일치하는 부분의 시작 부분 인덱스를 저장하는 배열 start_index를 동적 배열 할당한다.

```
matching = (int*)calloc(strlen(pattern), sizeof(int));
start_index = (int*)malloc(strlen(string) * sizeof(int));
```

패턴의 anagrams과 문자열의 일치 여부를 확인하는 과정은 다음과 같다.

```
for (i = 0; i <= strlen(string) - strlen(pattern); i++) {
                for (j = i; j < i + strlen(pattern); j++) {
                        for (k = 0; k < strlen(pattern); k++) {
                                if (string[j] == pattern[k] && matching[k] == 0) {
                                        matching[k] = 1;
                                }
                        }
                }
                for (k = 0; k < strlen(pattern); k++) {
                        if (matching[k] == 0) {
                                break;
                        }
                }
                if (k == strlen(pattern)) {
                        start_index[count++] = i;
                }
                for (k = 0; k < strlen(pattern); k++) {
                        matching[k] = 0;
                }
        }
```

첫번째 반복문은 패턴의 길이만큼 검사할 문자열의 첫부분의 인덱스를 가리키는 i를 설정하는 부분이다. 이를 통해 문자열의 첫번째 문자부터 시작하여 문자열의 길이에서 패턴의 길이를 뺀 검사가 가능한 문자열의 마지막 부분까지 반복이 진행된다. 두번째 반복문은 문자열의 i번째부터 패턴을 길이만큼 문자열의 문자를 가리키는 인덱스를 지정한다. 이후 내부의 반복문은 패턴내의 문자를 순차적으로 가리키는 인덱스를 의미하는데, 문자열과 패턴을 하나씩 대응해보며, 일치하면서 이전 문자열과는 대응되지 않은 패턴의 문자를 찾아낸다. 만약 이전 문자열의 문자와 대응되지 않고 현재 비교하고자 하는 문자열의 문자와 일치한다면 해당 패턴의 문자의 인덱스를 가리키는 matching 배열의 원소 값을 1로 변경한다. 모든 문자열에 대해 검사를 수행한다. 이후 반복문을 이용해 matching 배열에 0이 존재하는지 여부를 확인한다. 만약 0이 존재한다면 이는 1대1로 모든 문자열과 패턴이 대응되지 않은 것을 의미하므로 패턴의 모든 anagram과 문자열이 일치하지

않는다는 것을 의미한다. 따라서 break를 사용하여 검사를 중지하고, k 값의 증가를 멈춘다. 만약 마지막까지 0값이 발견되지 않았다면 이는 모든 패턴과 문자열의 문자가 일대일 대응이 되었음을 의미하므로 k값이 패턴의 길이와 같아지게 된다. 이는 패턴의 anagram중 하나가 문자열과 일치했음을 의미하고 일치하는 문자열의 첫번째 인덱스를 저장하는 배열 start_index의 배열에 인덱스를 저장한다. 이후 마지막 반복문을 통해 matching의 원소를 모두 0으로 초기화하여 다음 검색을 준비한다.

출력부분은 리스트 형태로 출력하기 위해 다음과 같이 구성된다.

```
printf("[");
for (i = 0; i < count; i++) {
        printf("%d", start_index[i]);
        if (i != count - 1) {
                printf(", ");
        }
}
printf("]₩n");
```

배열의 마지막 원소 이후에 ‘,’을 출력하지 않기 위해 조건문을 사용하였다.

마지막으로 동적 할당된 배열 matching과 start_index를 free를 사용하여 해제해준다.

```
free(matching);
free(start_index);
```

문제3의 구조도는 다음과 같다.